Final Project Report
SI 507

Introduction

Recommendation system has become a vital part of modern society. With the expansion of internet connectivity, people across the world scroll over billions of queries everyday. It becomes important to store and utilize these data. Recommendation system is one of the ways to analyze and utilize this vast amount of data to give suggestions for new users.
As part of the final project, I am implementing a classical recommendation system for jobs purely based on graph theories. Users will give attributes which are essential for recommendation, example - Specific keyword, category of job looking for, experience and location. More attributes can be added depending on requirement. But the current implementation only takes the above criteria into consideration.

Data Source

The database of jobs is created by scraping the data from multiple sources. A total of **430** jobs are extracted containing **7 attributes,** namely,

1) Job title,
2) Job URL
3) Job location
4) Company name
5) Job Description
6) Basic qualification
7) Preferred qualification.

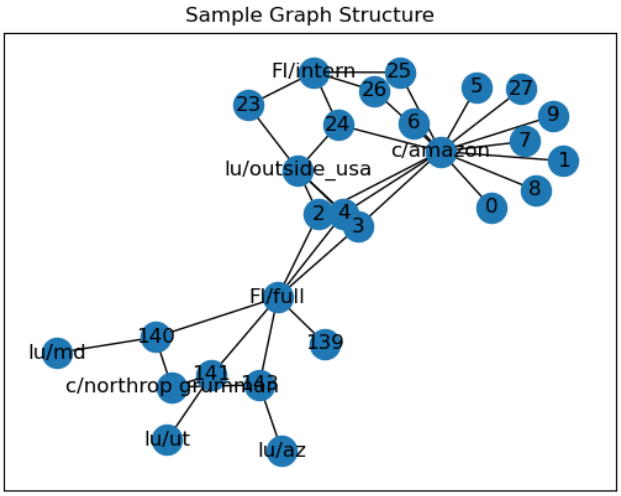
```
class JobPosting():
    def __init__(self, job_title, company_name, location, job_url, job_description=None,
                 basic_qualificaiton = None,
                 preferred_qualification = None):
        self.job_title = job_title
        self.company_name = company_name
        self.location = location
        self.job_url = job_url
        self.job_desc = job_description
        self.base_qual = basic_qualificaiton
        self.pref_qual = preferred_qualification

    def print_info(self):
        print("Job title :-", self.job_title)
        print("Company Name :-", self.company_name)
        print("Location :-", self.location)
```

These jobs are taken in such a way to cover a wide range of interest, location, and experience to give users enough options to explore. The data is extracted from 4 different sources, three are company websites such as Amazon jobs, Microsoft Jobs, and Apple jobs, whereas one is career website i.e Glassdoor. Crawling and scraping of multiple pages from the above websites are done to construct the total of 430 jobs. There are no inbuilt APIs for us to use for this website, so the scraping of jobs is done using BeautifulSoap and Selenium library, which helps in extracting information from the HTML directly. After extracting, the information is cached in a

local system to make sure the user doesn't waste time extracting the information everytime they make a query request. The caching information is dumped in a JSON file for it to use later.
The figure above shows the class which is used to wrap up the information about each job posting.

Data Structure

The database of jobs are stored in Graph data structure where each children node corresponds to job posting and each parent node corresponds to criteria that the user wants to look for, For example, the user wants to search for jobs based on location, then the locations are the parent node and all the jobs which are readily available in that location will be connected to this parent node. Conveniently one can assume to have a vast graph connectivity since the number of jobs are quite high and the criterias that the user can select also offer a reasonable amount of choices. Each child node stores two things, key and value. Key here refers to a unique identifier for the node and the value refers to information regarding jobs such as Job title, Job URL and so on (basically the 7 attributes that are discussed in the above section.). Each parent node on the other hand also stores similar information. For visualization purposes, I took a few nodes to show the sample connectivity of how graphs are connected.



In the above graph, we could see some nodes have very high connectivity (Such as "c/amazon"), they are most likely to be parent nodes and are connected to many child nodes satisfying certain conditions. The children nodes are terminal nodes and most likely have one way directed connection. Below shows some code snippets describing formation of graphs, and connection of nodes.

```python
class Graph:
    def __init__(self):
        self.adj_list = {}
        self.node_info = {}

    def add_edge(self, node1, node2): # node1 --> node2
        try:
            self.adj_list[node1].append(node2)
        except:
            self.adj_list[node1] = [node2]

    def add_node(self, key, val):
        self.node_info[key] = val

    def print_graph(self):
        for key, val in self.adj_list.items():
            print("Node {} is connected to {}".format(key, val))
```

Three code snippets, describing the construction of graphs, adding of nodes to the graph and making the connection of different nodes in the graph. The adjacency list is handled through a dictionary data structure describing the connections one node can have with other nodes.

```
--------------------------------------------------
Job title :- Director, Applied Science - PXTCS, PXT Central Science
Company Name :- Amazon
Location :- USA, VA, Arlington
Job Description :- The Central Science Team within Amazon's People Experience and Technology org (PXTCS) uses economics
Basic Qualification :- • Advanced degree (e.g., M.S. or PhD) in computer science, statistics, engineering, mathematics o
r r
Preferred Qualification :- • Ability to work in a fast-paced business environment.• Experience implementing algorithms,
tailore
--------------------------------------------------
Job title :- Manager, Applied Science (Machine Learning)
Company Name :- Amazon
Location :- GBR, London
Job Description :- Job summaryAre you interested in building state-of-the-art machine learning systems for the most com
Basic Qualification :- PhD in Computer Science, AI, Mathematics, or Statistics with specialization in ML (alternatively,
 MS
Preferred Qualification :- 3+ years of experience, and strong expertise, in any of: Probabilistic Machine Learning, time
 series
--------------------------------------------------
Job title :- Senior Applied Science Manager
Company Name :- Amazon
Location :- AUS, SA, ADELAIDE
Job Description :- The Amazon ML AU team is developing state-of-the-art, large-scale Machine Learning methods and appli
Basic Qualification :-
Preferred Qualification :-
--------------------------------------------------
```

```python
job_graph = Graph()

# Parent Node --> Location, Full/Intern, Company Name, Keyword
Nparent = 4
parent_node = ['Location', 'Full/Intern', 'Company Name', 'Keyword']

# Add all the node in job_graph
for idx, val in enumerate(PostList):
    job_graph.add_node(idx, val)

loc_dict_us, loc_dict_outside = get_unique_loc(PostList)
cmp_dict = get_unique_company(PostList)

# Add Parent Node with special keys to recognize it later.
for idx, (key, val) in enumerate(cmp_dict.items()):
    job_graph.add_node('c/'+str(key), key)

for idx, (key, val) in enumerate(loc_dict_us.items()):
    job_graph.add_node('lu/'+str(key), key)

job_graph.add_node('lu/outside_usa', 'outside_usa')

job_graph.add_node('FI/full', 'full')
job_graph.add_node('FI/intern', 'intern')
```

```python
# Connect all the edges based on location
for idx1, (key1, val1) in enumerate(job_graph.node_info.items()):
    if(isinstance(key1, int)):
        continue
    parent = key1.split('/')[0]
    for idx2, (key2, val2) in enumerate(job_graph.node_info.items()):
        if(parent in ['c'] and isinstance(key2, int)):
            if(val1.lower() == val2.company_name.lower()):
                job_graph.add_edge(key1, key2)
                job_graph.add_edge(key2, key1)

        if(isinstance(key2, int)):
            if(parent in ['lu', 'lo']):
                if(val2.location.lower()[:3]=='usa'):
                    if(val1.lower() == val2.location.lower().split(',')[-1].strip()):
                        job_graph.add_edge(key1, key2)
                        job_graph.add_edge(key2, key1)
                elif (len(val1)>=4) and (val1 not in ['virtual','remote']):
                    job_graph.add_edge('lu/outside_usa', key2)
                    job_graph.add_edge(key2, 'lu/outside_usa')

        if(isinstance(key2, int)):
            if(parent in ['FI']):
                ispresent = (val1=='intern') and (val1 in val2.job_title.lower())
                if(ispresent):
                    job_graph.add_edge('FI/intern', key2)
                    job_graph.add_edge(key2, 'FI/intern')
                elif(val1=='full' and ('intern' not in val2.job_title.lower())):
                    job_graph.add_edge('FI/full', key2)
                    job_graph.add_edge(key2, 'FI/full')
```

Interaction and Presentation Options

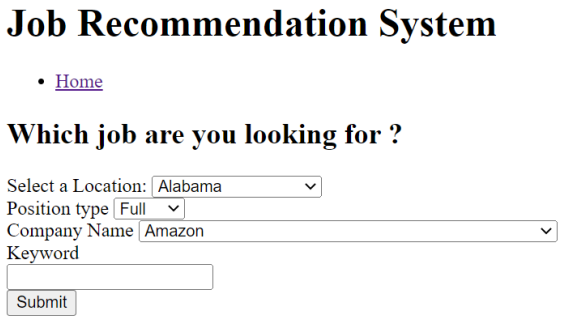
An interactive environment is created for the user to interact with the program directly on the web browser. The API allows users to enter certain requirements which they are looking for in the jobs. The user interface is built on flask. I wrote an HTML, and a simple Django script. In backend the program iterate over the graphs following Depth-first search to find the jobs satisfying the user conditions and finally displaying the results on the web browser allowing user to see general information about the Jobs such as title, location, company name, and URL which user can click to divert the page to the company website where they can apply directly.

Users on the web browser have the option to select the location they want to work on, these locations are primarily states in the US, also giving users the option to look for jobs outside the US. Users can also select, type of employment, are they looking for full time positions or Internships. The user interface also allows users to select companies from the drop down list, giving them options to select their preferred company or they can select all if they want to explore all possible jobs. Additionally, the user interface also allows the user to search for specific keywords in the jobs, this feature bridges the gap and gives users a chance to look for all possible jobs following a specific keyword irrespective of other constraints. In short, there are the following options available for users to interact.

1. A drop down menu for users to select location.
2. Type of employment, Full time or internship.
3. A drop down menu for users to select companies.
4. Keywords.

Instructions for the UI :-

1. Fig 1 shows the Home page, it lets users give input such as choosing a location, type of employment, company name, and keywords.
2. After entering the input, the user can click on the submit button.
3. Fig 2 shows after submitting, results will be displayed letting the user interact with it further, either by clicking on the Job URL, or selecting other choices.
4. Users can keep interacting with the UI until satisfied.



Project Code

All the required python script and cache file are pushed in the github repository and can be found out here. The repository has a README file explaining how to operate the program, what each file means and how to download the libraries essential for the program to run. I have used *Flask* for generating a better looking UI for the users to interact conveniently. Below packages are other essentials required for the code to execute.

- BeautifulSoup
- requests
- pandas
- numpy
- selenium
- webdriver_manager
- contextlib
- Flask

In order to run the program simply call `python main.py` in the terminal or simply run `main.py` in your appropriate python editor. Then the terminal will pop up a local URL which can be used to interact with a program on a web browser directly. The web browser will let users decide on what kind of job they are looking for, based on the query, the python script will generate suggestions satisfying the user requirements. The user will be able to see general information about the Jobs such as title, location, company name, and URL which the user can click to divert the page to the company website where they can apply directly.

Demo

Demo video can be found out here